DSI: AI Development Program

Part 1: Introduction to AI Development

University of Chicago

January 14, 2026

Outline

1. Motivation and Learning Objectives
2. Foundations & Terminology
3. Context
4. Pricing
5. Exercise

Who Am I?

- Director Data Science Clinic
- Undergrad: UC Berkeley, PhD: UCLA
- Worked in Consulting, Video Games, AI
- I'm also an investor and consultant for AI Startups.

Some Games I've Worked on

Outline

1. Motivation and Learning Objectives
2. Foundations & Terminology
3. Context
4. Pricing
5. Exercise

Motivation: Why AI Development?

- AI can be used directly through chat interfaces (ChatGPT, Claude)
- However, most production AI systems are built **programmatically** through APIs
- Why?
  1. **Control**: Cost management, output formatting, verification
  2. **Integration**: Connect AI to existing systems (databases, CRMs, APIs)
  3. **Human Oversight**: Code provides guardrails against hallucination
  4. **Automation**: Scale beyond manual chat interactions
  5. **Scalability**: Building systems that can scale beyond a one-to-one human-chat.

Example: Personalized Marketing Outreach

- A company wants to use their CRM system to send personalized cold outreach emails
- They need to write code to:
  - Connect to their CRM database
  - Retrieve prospect information
  - Generate personalized emails using an LLM API
  - Send emails through their email system
- This requires **programmatic integration**, not just chat interfaces
- This is a concrete example of why we build AI systems programmatically

Course Learning Objectives

By the end of this course, you will:

1. **Understand AI terminology**
    - Don't make an ass out of yourself during an interview
    - Speak confidently about LLMs, tokens, context windows, agents, tools
2. **Gain familiarity with cost drivers**
    - Understand what makes ROI sense and what doesn't
    - Calculate and optimize token costs
3. **Experience building AI systems**
    - Have concrete experiences to talk about in interviews
    - Tune and optimize real systems
    - Understand the trade-offs in building such systems

Today: Workshop 1

- Focus on the **basic** definitions and how to work with these models
- Build foundation for hands-on work
- About 1 hour of lecture, 30 minutes hands-on

Next lectures:

- Workshop 2 - 4: Specific Implementation that highlight concrete problems
- Dates: Thursday (1/15), Monday (1/26) and Tuesday (1/27)
- Workshops will be 30-ish minutes of lecture followed by 1 hour of hands-on time
- Between each lecture there will be a set of readings (usually articles)

Outline

1 Motivation and Learning Objectives

2 Foundations & Terminology

3 Context

4 Pricing

5 Exercise

So what is AI?

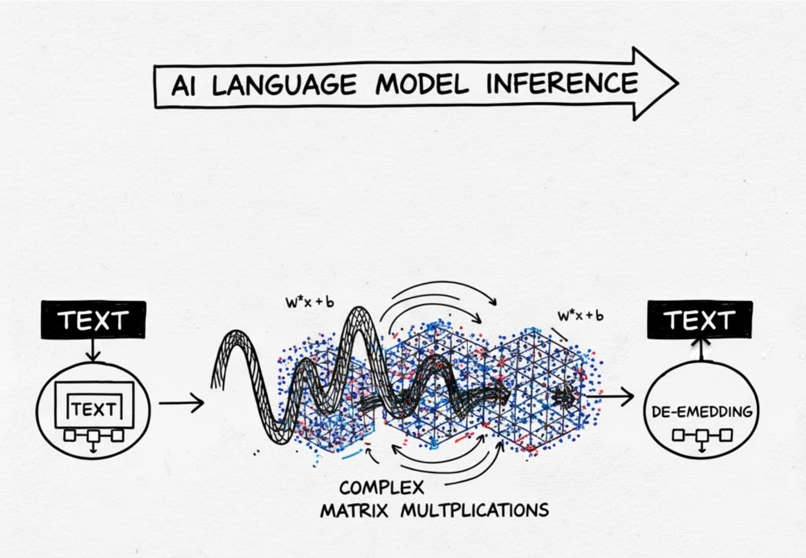
When people say “AI” today, they usually mean **Large Language Models (LLMs)** or **Foundation Model**

… So what is an LLM / Foundation Model?

Lets start with LLMs!

- A **Large Language Model (LLM)** is an artificial intelligence system trained on massive amounts of text data (often trillions of words)
- These models learn patterns in language and can:
  - Generate human-like text
  - Answer questions
  - Translate languages
  - Summarize documents
  - Write code
  - And much more
- Think of an LLM as a very sophisticated autocomplete system that has read a significant portion of the internet

LLM Mental Model: Simplified



Foundation Models

- What about images and text?
- A **foundation model** is a large-scale machine learning model trained on broad data
- **Key characteristics:**
  - Broad training: Trained on diverse, large-scale datasets
  - General-purpose: Can be adapted to many different tasks
  - Transfer learning: Knowledge learned from training can be applied to new tasks
  - Base for specialization: Can be fine-tuned or used as-is for specific applications

Foundation Models vs LLMs

- **Foundation Model** is the broader category
  - Can handle multiple modalities (text, images, audio, video)
  - Often called “multimodal models”
- **LLM** (should) specifically refers to language models
- **Colloquially**: People often say “LLM” to mean “foundation model”
- They operate the same way (embedding → transformation → output)

Examples of foundation models:

- **Language**: GPT-4, Claude, Llama (these are LLMs)
- **Vision**: CLIP, DALL-E
- **Multimodal**: GPT-4V (vision + language), Gemini

Key Terminology

- **Token**: The basic unit of text that an LLM processes (word, part of word, or punctuation)
  - 1 token $\approx$ .75 words (so a bit less than a word)
- **Prompt**: The input text you give to an LLM
- **Completion/Response**: The output text generated by the LLM

How Do We Use These Models?

1. **Chat Interfaces**: ChatGPT, Claude, Gemini
   - Direct user interaction
   - Good for exploration and prototyping
2. **Terminal Interfaces**: Claude Code CLI, Open AI Codex, OpenCode
   - Command-line access
   - Integration with development workflows
3. **API (Application Programming Interface)**
   - **This is our focus** - programmatic access
   - Build production systems
   - Integrate with existing applications

LLM Limitations

While LLMs are powerful, they have important limitations:

1. **Knowledge Cutoff**: They only know information from their training data up to a certain date
2. **Hallucination**: They can generate plausible-sounding but incorrect information
3. **No Real-World Actions**: They can't directly interact with external systems (databases, APIs, file systems)
4. **Context Limits**: They have maximum context window sizes
5. **Static Knowledge**: They can't learn new information after training without fine-tuning or retrieval

Outline

1. Motivation and Learning Objectives
2. Foundations & Terminology
3. Context
4. Pricing
5. Exercise

Content Engineering: The Core of AI System Development

- Most of what we do when programming AI systems is **engineering what goes into the model**
- We call this **content engineering** (or **context engineering**)
- It focuses on manipulating the **context window** to generate responses that are most likely to succeed
- This is where the real work happens:
  - Designing effective prompts
  - Structuring information for the model
  - Providing the right examples and constraints
  - Managing context size and relevance
- **The model is powerful, but what you put in determines what you get out**

Context windows: What they are

- The **context window** is the model's working memory for a single request
- Everything the model can use must fit inside it
- When you exceed the window, something gets dropped or truncated
    - Some tools will do automatic context window management (compacting, etc), but the trade-offs remain
- So how big are these context windows?

Context Window Size Limitations

- Different models have different maximum context window sizes (measured in **tokens**)
- Common model context windows (as of 2024-2025):
  - **GPT-4**: 128,000 tokens
  - **Claude 3.5 Sonnet**: 200,000 tokens
  - **Claude Sonnet 4**: 1,000,000 tokens (announced 2025)
  - **Llama 3.1** (70B, 405B): 128,000 tokens
- **Remember**: 1 token $\approx$ 0.75 words
- Larger context windows enable:
  - Processing longer documents
  - Maintaining longer conversation history
  - Including more retrieved information
- But they also increase cost and latency

Context windows: practical implications

- **Tradeoffs**: More context can improve grounding, but increases cost/latency
- **Recency and placement matter**: Newer text is usually attended to more

Breaking Down the Context Window

- For commercial LLMs, the context window can be broken into 3 components
  1. **System Prompt**: High-level instructions & safety rules
  2. **User Prompt**: The actual request or question
  3. **Environment Context**: Retrieved documents, tool outputs, conversation history
- Understanding this breakdown helps with:
  - Cost optimization (what's necessary vs. optional)
  - Prompt design (where to put different types of instructions)
  - Debugging (what the model actually sees)

System Prompt

- The **system prompt** is separate from the user prompt and sets the model's behavior
- The **system prompt** sets the highest-level behavior (tone, rules, safety, format)
- It is NOT under our control
- It can be changed without us noticing or warning

See example system prompts: `https://github.com/guy915/System-Prompts`

Environment Context

- We can add additional features to our context window to give "abilities" to our LLM
- For example: we could add a tool that would allow our LLM to browse our hard drive (in the case of local Claude code)
- There are a lot of different patterns for this:
  - MCP
  - Skills
  - Tools
  - `AGENTS.md` files
- Anything programmatically accessible that helps the model understand the environment, task, or user's situation
- These are generally put into the context window one time and then always exist when starting up.

User Prompt

- The **User Prompt** or **User Messages** is what we provide the LLM
- This is our most common interaction point

Context Failure

- When LLMs fail it is (frequently) due to problems by incorrectly engineered context.
- **Failure modes** to watch for:
  - forgetting earlier constraints
  - losing critical details due to truncation
  - quoting the wrong source when too many docs are included

Outline

1. Motivation and Learning Objectives
2. Foundations & Terminology
3. Context
4. Pricing
5. Exercise

Token economics: why tokens matter

- Most LLM APIs charge by tokens:
  - **input tokens** (prompt, system prompt, retrieved docs, tool outputs)
  - **output tokens** (the model's response)
- Cost and latency generally scale with total tokens processed
- **Context engineering is also cost engineering**:
  - tighter prompts → lower cost and often higher reliability
  - smaller context when possible → faster, cheaper

Real-world pricing examples

Example (OpenAI GPT-4.1 family; per 1M tokens)

Model	Input	Output
GPT-5.2	$1.75	$14.00
GPT-4.1	$2.00	$8.00
GPT-4.1 mini	$0.40	$1.60
GPT-4.1 nano	$0.10	$0.40

Source: `https://platform.openai.com/docs/pricing`

Cost math (simple model)

- Typical pricing is “$ per 1M tokens”
- Approximate cost per call:

$$\text{cost} \approx \left(\frac{T_{in}}{10^6} \cdot p_{in}\right) + \left(\frac{T_{out}}{10^6} \cdot p_{out}\right)$$

- Engineering levers:
  - reduce retrieved context (better retrieval, smaller snippets)
  - constrain output length (max tokens, structured output)
  - pick the cheapest model that meets quality requirements
  - cache repeated context (where supported)

Pricing references: `https://openai.com/api/pricing/`

Outline

1. Motivation and Learning Objectives
2. Foundations & Terminology
3. Context
4. Pricing
5. Exercise

Exercise

The goal for today is:

- Team up with another person (min group 2, max group 3)
- Get access to your API Keys
- Verify that you can use the keys
- Complete the notebook. Can you engineer the context to get all the webpages to be classified correctly?